VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2017 Adam Podlas

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Adam Podlas**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Jan Platoš, Ph.D.**

Konzultant bakalářské práce: Mgr. Zdeněk Dřizga

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * 1D16 * Ostrava *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 6. dubna 2017 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

Tieto Czech s.r.o.
28. října 3346/9
702 00 Ostrava - Moravská Ostrava
IČO 64808051 DIČ CZ64808051

V Ostravě 6. dubna 2017

Rád bych poděkoval firmě Tieto Czech s.r.o. za umožnění absolvování odborné praxe. Především pak mému konzultantovi Mgr. Zdeňku Dřizgovi. Dále chci poděkovat mému vedoucímu bakalářské práce doc. Ing. Jan Platoš, Ph.D.

Abstrakt

Tato práce popisuje průběh mé odborné praxe ve firmě Tieto Czech s.r.o. Zde jsem měl možnost vyzkoušet si práci na pozici Software developer a podílet se tak na vývoji dále popisovaných webových aplikací. Hlavní část práce se věnuje konkrétním úkolům, které mi byly v průběhu praxe zadány, a jejich řešením. V závěru práce hodnotím celkový přínos praxe, jaké teoretické znalosti nabyté ve škole mi pomohly a jaké naopak scházely.

Klíčová slova: Odborná praxe, Tieto Czech, Webové aplikace, Java, JavaScript, React JS

Abstract

This paper describes the course of my professional experience in company Tieto Czech Ltd. Here I had the opportunity to work on the position of software developer and contribute to the development of web applications described below. The main part is dedicated to specific tasks, which were assigned to me during practice, and their solutions. In conclusion I evaluate the overall benefit of practice, what theoretical knowledge acquired in school helped me and, which I lacked.

Key words: Proffesional practice, Tieto Czech, Web applications, Java, JavaScript, React JS

OBSAH

OBSAH 8
Seznam použitých zkratek a symbolů 9
Seznam obrázků 10
Seznam útržků zdrojového kódu 11
1. Úvod 12
2. O firmě 13
2.1. Odborné zaměření firmy 13
2.2. Pracovní pozice 13
3. Zadané úkoly 14
3.1. Place Reservations 2 14
3.2. CodeStorm 14
3.3. Tieto Area Info a Place Reservations 3 15
3.3.1. Tieto Area Info - Architektura 16
3.3.2. Tieto Area Info - frontend 17
3.3.3. Tieto Area Info - backend 22
3.3.4. Place Reservations 3 22
3.4. Utilizace vývojového prostředí a práce s Javascriptem 28
3.4.1. Našeptávač 28
3.4.2. Zpřehlednění kódu 29
3.4.3. Ladění 29
4. Uplatněné a chybějící znalosti 30
5. Závěr 31
6. Literatura 32

Seznam použitých zkratek a symbolů

ES	EcmaScript
REST	Representational state transfer
API	Application programming interface
SPA	Single page application
AJAX	Asynchronous JavaScript and XML
IDE	Integrated development environment
PNG	Portable Network Graphics
SVG	Scalable Vector Graphics
CRUD	Create, read, update, and delete
IOT	Internet of things
CLI	Commandline interface

Seznam obrázků

1) Hlavní stránka Place Reservations 2 14
2) Hlavní stránka CodeStorm 15
3) Zjednodušené rozložení jednotlivých modulů TAI 16
4) DevTools: Kompletní store aplikace 20
5) TAI mapa integrovaná do Place Reservations 3 s vlastními elementy 21
6) TAI editor 22
7) Place Reservations 3: Dashboard 24
8) Joyride v aplikaci Place Reservations 3 26

Seznam útržků zdrojového kódu

1) Kód 1 Ukázka stateless komponenty 18
2) Kód 2 Ukázka stateful komponenty 18
3) Kód 3 Ukázka syntaxe Redux akce 19
4) Kód 4 Metoda reducer měnící store aplikace 19
5) Kód 5 Ukázka syntaxe Joyride stepu 25
6) Kód 6 Metoda, která po akci na otevření dialogu spustí přísušné pole stepů 25
7) Kód 7 Metoda spojující data PR a TAI 27

1. Úvod

V této práci popisuji své působení ve firmě Tieto Czech s.r.o. v rámci odborné praxe. Do firmy jsem nastoupil již v lednu 2016, tedy na začátku mého letního semestru 2. ročníku. Hledal jsem způsob, jak se už v době studia seznámit s reálným pracovním procesem a studentská stáž se ukázala jako vhodné řešení. Tieto jsem si vybral na základě pozitivních recenzí studentů, kteří zde už stáž podstoupili. Po kontaktování společnosti a absolvování dvou kol pohovorů jsem byl tedy přijat na pozici software developer a přiřazen do týmu k dalším stážistům.

V následujících kapitolách práce uvádím popis firmy Tieto Czech s.r.o. a mou pozici ve firmě. Seznam úkolů, které mi v průběhu praxe byly zadány spolu s jejich časovou náročností a postupem jejich řešení. Dále uplatnění znalostí a dovedností získaných ve škole a naopak znalosti, které mi v průběhu praxe scházely. Na závěr pak shrnuji dosažené výsledky praxe a její celkové zhodnocení.

2. O firmě

Do České republiky společnost Tieto vstoupila v roce 2001 a v roce 2004 otevřela své softwarové centrum v Ostravě. S více než 2 200 zaměstnanci je jedním z největších zaměstnavatelů v oblasti IT služeb v České republice a největším v Moravskoslezském kraji. Z hlediska počtu kmenových zaměstnanců je česká pobočka třetí největší pobočkou Tieto korporace na světě. První dvě místa zaujímají mateřské země Finsko a Švédsko.

2.1. Odborné zaměření firmy

Tieto je největší severoevropská IT společnost poskytující komplexní IT servis. Zajišťuje také služby v oblasti vývoje produktů pro firmy působící v odvětví komunikací a integrovaných technologií. Na základě svých znalostí, technologických vizí a inovativního myšlení se společnost Tieto aktivně snaží inspirovat a zapojit své zákazníky do hledání nových způsobů, jak zefektivnit jejich podnikatelskou činnost.

Tieto spojuje globální možnosti s přítomností na lokálních trzích a zároveň staví na svém severském odkazu. Společnost Tieto se sídlem ve finských Helsinkách zaměstnává přes 13 000 expertů téměř než 20 zemích světa. Obrat činí přibližně 1,5 miliard eur. Akcie společnosti Tieto jsou obchodovány na burze NASDAQ v Helsinkách a Stockholmu. [1]

2.2. Pracovní pozice

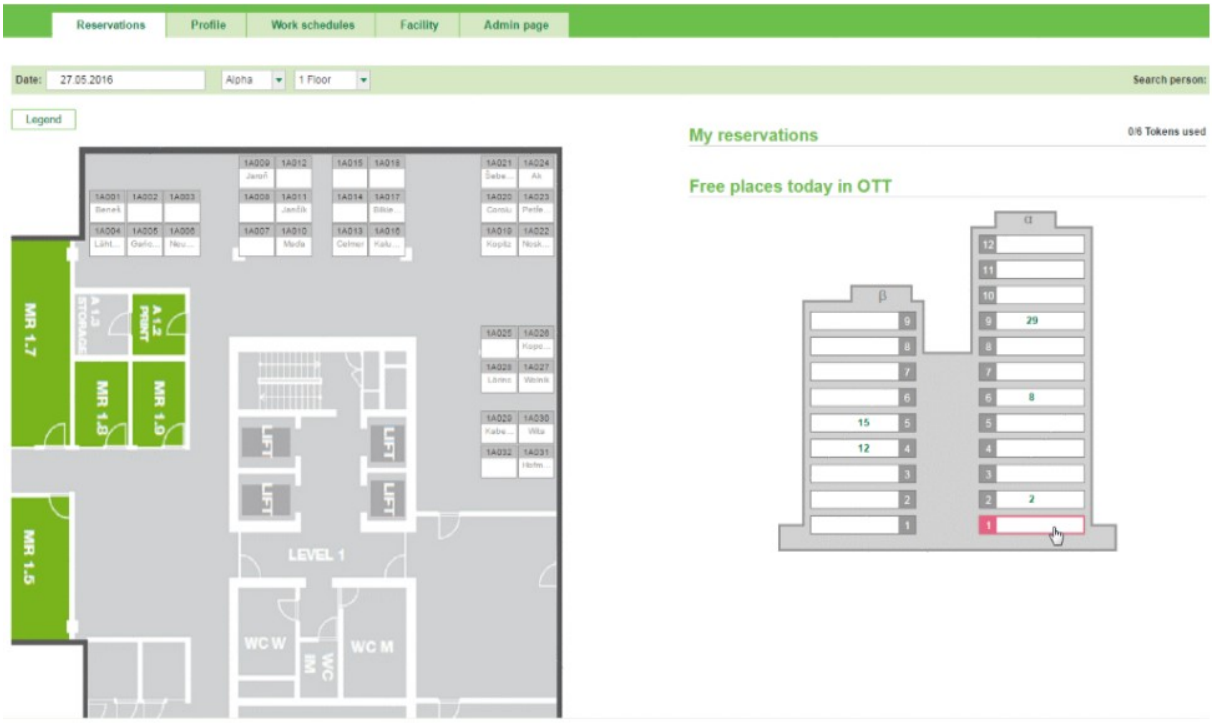
Do firmy jsem nastoupil na pozici software developer. Spolu s týmem dalších stážistů jsem začal pracovat na interních projektech. Tato práce spočívala v převzetí odpovědnosti a údržby již existujících a zároveň vývoj nových webových aplikací pro další zaměstnance firmy. Převzaté aplikace byly založeny na frameworku Spring v kombinaci s technologií JSF a knihovnou component Primefaces. Při vývoji nových jsme vsadili na moderní frontendovou Javascript knihovnu ReactJS, backend pak obstarávala odlehčená a zjednodušená verze Springu - Springboot. Z počátku jsem se zaměřoval především na vývoj backendu v jazyce Java, ale s postupem času jsem měl možnost poznávat mnohé moderní technologie a na konci praxe už jsem se věnoval téměř výhradně frontendové stránce aplikací. Konkrétně jazyku Javascript s knihovnou ReactJS.

3. Zadané úkoly

V průběhu své praxe jsem pracoval na několika úkolech. Z počátku má práce spočívala v údržbě již hotové interní webové aplikace Place Reservations 2 (dále jen PR). Následně jsem se podílel na vývoji aplikace Code Storm (dále jen CS). Převážnou část praxe jsem však strávil vývojem aplikací Tieto Area Info a nové verzi PR 3.

3.1. Place Reservations 2

Můj první zadaný úkol spočíval v převzetí funkční aplikace PR, která byla v té době užívána zaměstnanci firmy. Aplikace sloužila k rezervaci sdílených pracovních míst. To jsou místa, která nejsou trvale přidělena zaměstnancům, ale jsou každý den užívána kýmkoliv, kdo si vytvoří rezervaci. Po převzetí aplikace jsem měl za úkol implementovat drobné požadavky na změny. Aplikace běžela na frameworku Spring a frontend byl založen na technologii JSF. Na tomto projektu jsem pracoval asi 20 dní.

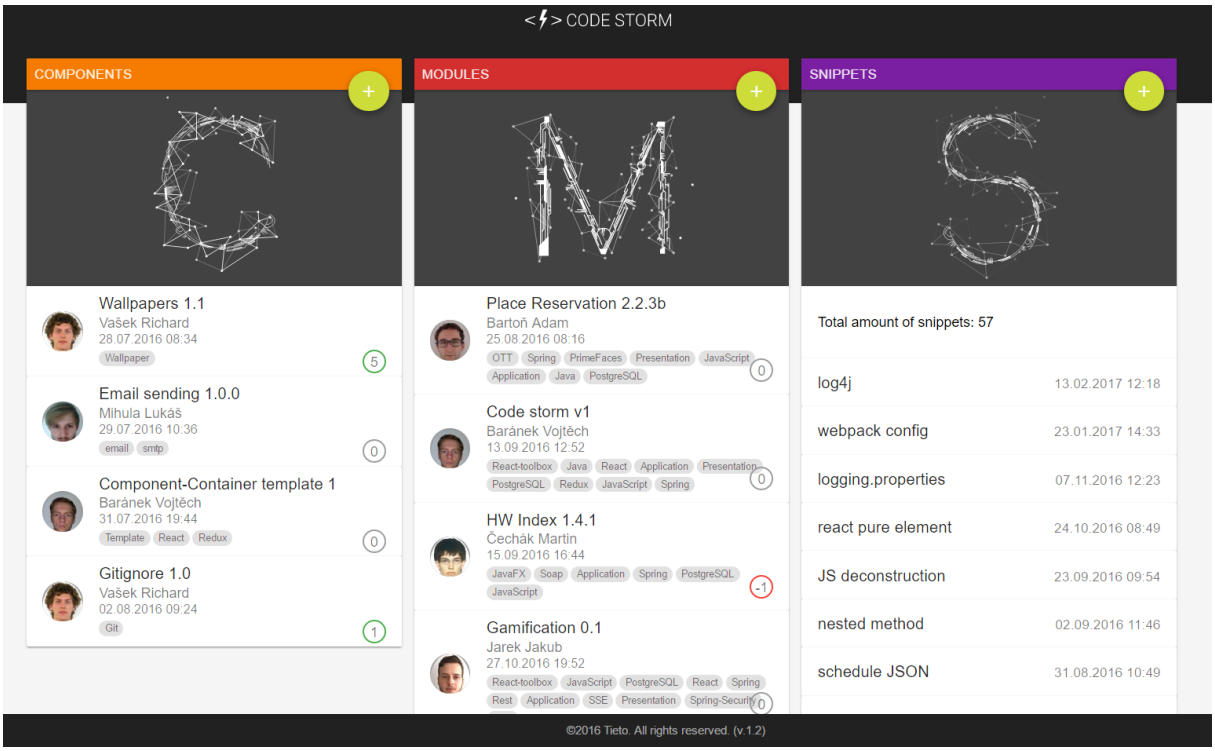


Obrázek 1 Hlavní stránka Place Reservations 2

3.2. CodeStorm

Tato aplikace slouží ke sdílení produktů práce napříč různými odděleními firmy. Systém obsahuje 3 moduly. Jeden pro hotové produkty, druhý pro znovupoužitelné části kódu. Třetí modul byl přídán později a slouží pouze pro rychlé uložení několika řádků kódu a následné vygenerování odkazu pro sdílení (oficiální firemní komunikační nástroj má problémy s kopírováním a posíláním částí zdrojového kódu).

Hlavní myšlenka CS byla vytvořit jakýsi interní katalog produktů firmy, aby se manažeři mohli před nákupem např. určitého monitorovacího nástroje podívat, zda už nějaké oddělení nevyvinulo vlastní podobný produkt, který by se dal použít. CS byla první aplikace, u které jsem měl možnost účastnit se vývoje od samotného počátku. Spolu se mnou na této aplikaci pracovali další 4 stážisti a měli jsme za úkol zajistit vše, od návrhu databázového modelu, přes architekturu až po použití vhodných technologií. Backend byl tvořen PostgreSQL databází, obsluhovanou frameworkem Springboot. Pro komunikaci s frontendem v Javascriptu s knihovnou ReactJS bylo vystaveno REST API. Na vývoji CS jsem pracoval asi 20 dní.



Obrázek 2 Hlavní stránka CodeStorm

3.3. Tieto Area Info a Place Reservations 3

Se zvyšujícím se počtem požadavků na vylepšení PR 2 a jejich náročností jsme se dostali do bodu, kdy už nebylo možné požadavky plnit se zachováním stávajícího systému v rozumné době. Proto jsme se rozhodli pro vývoj třetí verze od začátku.

Součástí původního PR navíc byla kompletní mapa budovy a pracovních míst. Ta byla vytvořena jako statický PNG obrázek půdorysu budovy, na který se pomocí HTML a Javascriptu vykreslovaly divy představující místa. Tento systém se ukázal jako nevhodný pro jakékoliv rozšíření funkcionality.

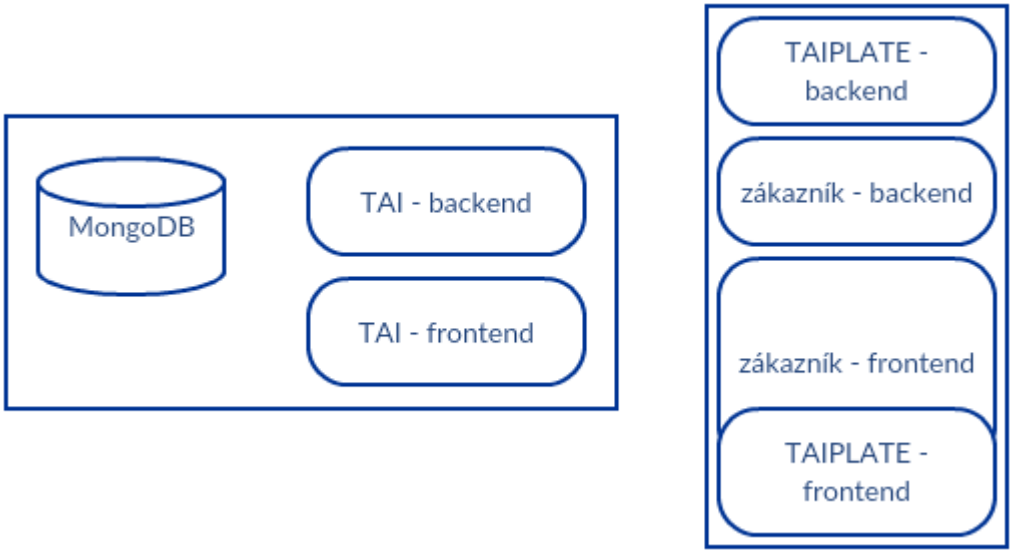
Abychom byli schopni novou verzi mapy vytvořit co nejuniverzálnější, zvolili jsme postup vytvoření mapového enginu jako samostatného projektu, který bude v podobě maven (backend) a npm (frontend) modulů možno přidat do jiných projektů. Tomuto mapovému enginu jsme dali pracovní název Tieto Area Info (dále jen TAI). PR 3 byla první aplikace, která měla TAI mapu používat. Vývoj těchto dvou projektů probíhal souběžně a zabral asi 50 dní.

3.3.1. Tieto Area Info - Architektura

Pro aplikaci jsme použili tří vrstvou architekturu a využívá vzoru microservices. Backend tvoří maven projekt v Javě a je postavený na frameworku Springboot. Jako úložiště dat používá NoSQL databázi MongoDB [2], se kterou komunikuje skrze framework Spring Data MongoDB. Frontend je napsaný v Javascriptu, konkrétně v nové verzi ES6, za použití knihovny ReactJS. Jelikož prohlížečům zatím chybí podpora ES6, překládá se tento kód pomocí nástroje babel do starší ES5. Jako runtime prostředí pro JS část aplikace slouží Node.js [3]. Pro komunikaci mezi frontend a backend částí se využívá REST API.

Celá aplikace je rozdělena na dvě části. Klientská část (dále jen TAIPLATE), která klientům dovolí prohlížet předem připravené mapové podklady a administrátorská část (dále jen TAI), která umožňuje nahrávat pozadí a pozicovat vrstvy objektů k zobrazení. TAI (MongoDB, TAI maven modul, TAI node.js aplikace) běží na našem centrálním serveru. Data jednotlivých zákazníku jsou potom rozděleny pomocí API klíče, který jim je přidělen při implementaci TAIPLATE.

Naproti tomu TAIPLATE je zakomponovaná do aplikace zákazníka. Frontendovou část si zákazník pouze importuje jako npm modul do své ReactJS aplikace. Backendová část, tedy TAIPLATE maven modul, je však nutno individuálně upravit pro potřeby každého zákazníka a upravenou verzi mu dodat. Tato úprava je nutná, protože TAIPLATE maven modul slouží ke sloučení dat z naší TAI databáze (umístění objektu, jeho velikost, orientace atd.) a zákazníkovy databáze.



Obrázek 3 Zjednodušené rozložení jednotlivých modulů TAI

3.3.2. Tieto Area Info - frontend

React

Základ frontendu aplikace tvoří již zmíněná javascriptová knihovna ReactJS. Hlavní myšlenkou této knihovny je umožnit „skládání“ webových stránek z nezávislých komponent. Při tom kombinujeme komponenty vlastní s těmi, jež vytvořil někdo jiný a zpřístupnil je ve veřejném npm repozitáři. Jelikož se jedná o SPA, vznikne skládáním jeden strom komponent. Při aktualizaci stavu aplikace se poté vždy znovu vykreslí pouze ta část stromu, kterou změna stavu ovlivní.

V Reactu existuje pouze *one-way data binding,* to v praxi znamená, že data se ve stromu komponent předávají pouze shora dolů. Žádná komponenta tedy nemůže upravovat data která ji jsou předána. Pokud takovou změnu dat chceme komponentě umožnit, musíme jí předat funkci, která data aktualizuje přímo v rodičovské komponentě.

Pro zápis React komponent se používá rozšíření syntaxe Javascriptu zvaná JSX. Toto rozšíření umožňuje zapisovat komponenty podobně jako HTML a pracovat s nimi jako s jakýmkoli jiným objektem – můžou sloužit k naplnění proměnné, nebo třeba tvořit vstup nebo výstup funkce. JSX také dovoluje vložení bloku Javascriptového kódu prakticky do libovolného místa v JSX elementu.

V Reactu existují dva základní typy komponent – *stateless function* a *stateful class. Stateless* komponenta je prostě jen funkce, jejíž vstupní argumenty jsou *props* a vrací JSX element. Jak už název napovídá, *stateless* komponenty nemají žádný stav. Například když se jedná o jednoduchý input, je třeba jeho hodnotu ukládat vně komponenty a předávat jí pomocí *props.* V praxi to znamená, že do každého takového inputu je třeba předat minimálně dvě položky: aktuální hodnotu a funkci, která tuto hodnotu změní. *Stateless* komponenty také postrádají možnost rozšíření metod životního cyklu jako *componentWillMount, componentDidUpdate,* atd. Odměnou za tyto nedostatky je jednodušší vnitřní implementace a tím pádem zvýšení výkonu. *Stateful* komponenty jsou zapsány jako třída rozšiřující třídu *React.Component.* V implementaci třídy existuje jediná podmínka – třída musí obsahovat metodu *render(),* která nemá vstupní argumenty a vrací JSX element. Jak k *props,* tak ke *state,* přistupuje tato třída přes klíčové slovo *this.* [4]

```
export const DayCheckbox = ({ checked, onChange, disabled, day, date })
=> (
  <div
    className={`${/Sat|Sun/.test(day) ? styles.dayCheckboxWeekend :''}`}
  >
    <ReactTooltip id={date} effect={'solid'}>
      <span>
        {this.props.disabled
          ? 'Place is not free in this interval'
          : 'Reserve place for this day'}
      </span>
    </ReactTooltip>
    <div data-for={date} data-tip>
      <Checkbox
        className={styles.checkbox}
        checked={checked}
        onChange={() => onChange(this.props.date)}
        disabled={disabled}
      />
    </div>
    <div className={styles.label}>{day}</div>
    <div className={styles.label}>
        {moment(date).format('DD. MM')}
        </div>
  </div>
);
```

Kód 1 Ukázka stateless komponenty

```
class ProjectList extends Component {

  componentWillMount() {
    this.props.actions.fetchAllProjects();
  }

  render() {
    const { projects } = this.props;
    return (
      <div className={styles.projectList}>
        {
          projects ?
            projects.get('projects').toIndexedSeq().map(project =>
              <div className={styles.projectView}
                key={project.get('name')}
              >
                <ProjectView
                  project={project}
                  edit={this.props.actions.editProject}
                /></div>) : null
        }
      </div>
    );
  }
}
```

Kód 2 Ukázka stateful komponenty

Redux

Redux poskytuje velmi využívané řešení kontrolování stavu ReactJS aplikace. Jedná se v podstatě o kontejner (dále jen *store*), obsahující kompletní současný stav aplikace. K aktualizaci dat ve *storu* používáme akce. Zavolání akce je jediný způsob, jak změnit stav *storu*. Po zavolání akce se provádí metoda *reducer*. Vstupní parametry této metody jsou předchozí stav a volaná akce, výstup tvoří nový stav. *Reducer* tedy definuje, jakým způsobem dané akce změní stav aplikace

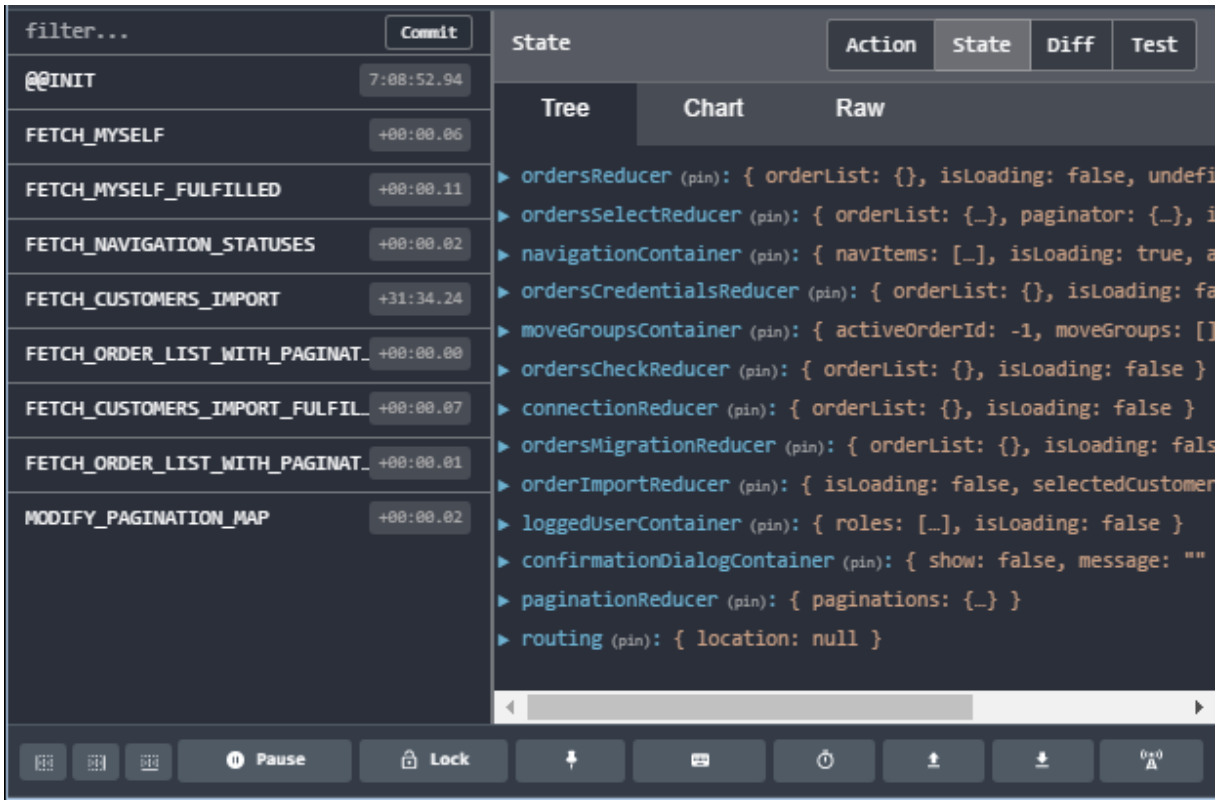
```
export const setSteps = (steps) => ({
  type: JOYRIDE_ACTION_SET_STEPS,
  steps,
});
```

Kód 3 Ukázka syntaxe Redux akce

```
const joyrideCustomReducer =
  (state = new InitialState(), action) => {
    switch (action.type) {
      case JOYRIDE_ACTION_SET_READY:
        return state.set('ready', action.ready);
      case JOYRIDE_ACTION_SET_STEPS:
        return state.set('steps', action.steps);
      default:
        return state;
    }
  };
```

Kód 4 Metoda reducer měnící store aplikace

Mezi hlavní výhody tohoto přístupu patří velmi snadné testování – pro každou akci lze snadno vygenerovat test, který zkontroluje, zda se po zavolání akce správně upravil stav. Další výhodou je jednoduché ladění. S pomocí např. Redux dev tools můžeme zvrátit již provedené akce a sledovat, jak jednotlivé akce modifikují stav aplikace – tzv. *time traveling.* [5]



Obrázek 4 DevTools: Kompletní store aplikace

Redux Observable

Redux Observable uplatňuje reaktivní paradigma na Redux akce. Díky tomu je možné vytvářet z akcí streamy a aplikovat na ně operátory typické pro reaktivní programování – např. *throttle, debounce, takeUntil*, atd. Toto je klíčová součást našich aplikací, jelikož s její pomocí řešíme mimo jiné i volání REST API (vybrané Redux akce se ve streamu mapují na *ajax* požadavky. [6]

TAIPLATE

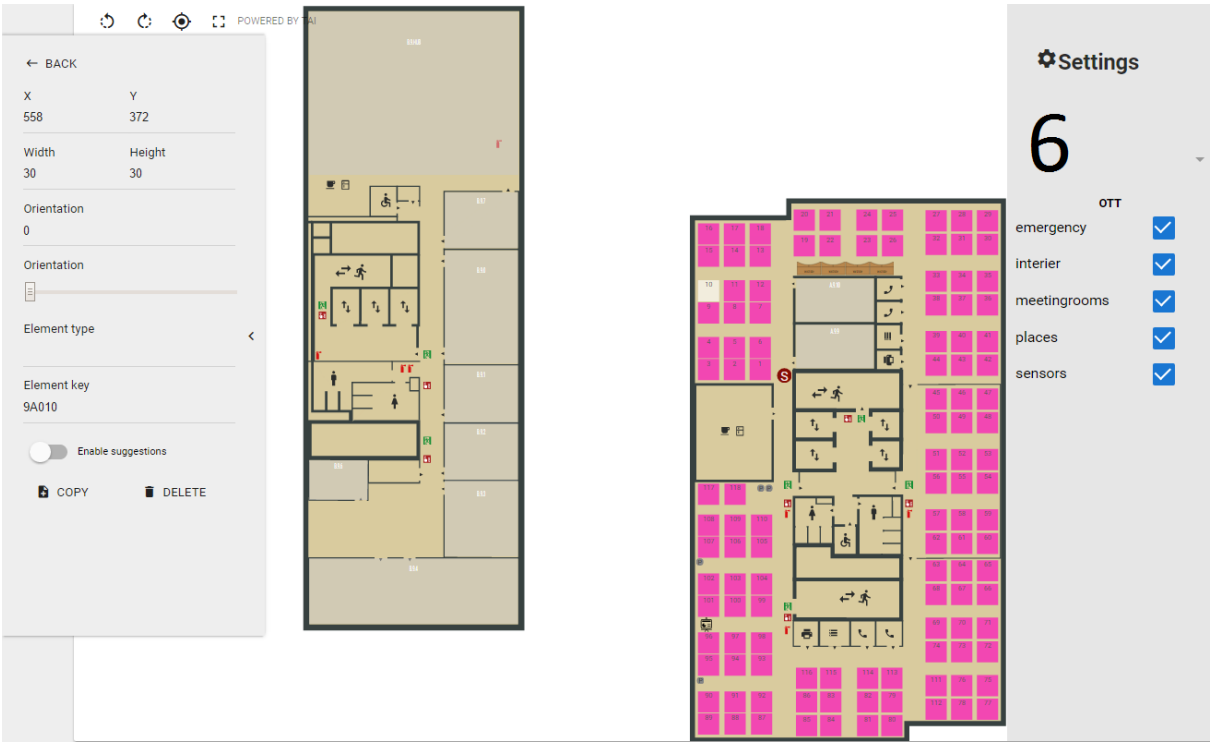
Klientská část obsahuje pouze samotnou mapu. Na pravé straně mapy se nachází ovládací panel. Zde si uživatel vybere jedno ze seznamu dostupných pozadí. Po vybrání pozadí se zde objeví seznam dostupných vrstev objektů pro toto pozadí. Uživatel si může současně zobrazit libovolný počet těchto vrstev. V levém horním rohu mapy se nachází tlačítka pro kontrolu mapy. Zleva jsou to: rotace o 90° vlevo, rotace o 90° vpravo, vycentrování mapy a režim celé obrazovky. Zbytek ovládání je typický pro mapové aplikace – přiblížení pomocí scrollování, pohyb po mapě tažením. Celá mapa je tvořena vektorově, takže úroveň přiblížení je neomezená.

Vývojáři aplikace využívající modul TAI mají mnoho možností definování uživatelských interakcí s objekty na mapě. Mohou jim nastavit vlastní vzhled, chování po kliknutí, nebo třeba nastavit vlastní komponentu, která se zobrazí po najetí myší na daný objekt – tzv. tooltip.

Obrázek 5 TAI mapa integrovaná do Place Reservations 3 s vlastními elementy

TAI

První krok při použití TAI editoru je nahrání pozadí. Předpokládá se, že většinou se bude jednat o půdorys budovy, ale uživatel v tomto směru není omezován. Kvůli zachování kvality mapy je nutné, aby bylo pozadí ve formátu SVG. Po úspěšném nahrání pozadí přichází vytvoření nové vrstvy. Vrstvy slouží k definování skupiny objektů, které budou vždy viditelné pouze současně. Vrstva je vždy vázána na jedno pozadí. K vytvoření vrstvy stačí zadat její název. Následně už můžeme do vrstvy přidávat libovolné objekty. [7]



Obrázek 6 TAI editor

3.3.3. Tieto Area Info - backend

TAIPLATE

Backendová část klientské aplikace je jediná, která se musí přizpůsobit pro každého zákazníka individuálně. Funguje jako spojení zákazníkových dat s daty mapovými. Toto spojení probíhá na servisní vrstvě modulu TAIPLATE. Pro identifikaci objektů se používá unikátní klíč, který zákazník v TAI editoru přiřadí objektům při jejich vytvoření. Jako tento klíč může sloužit například primární klíč odpovídající objektu ze zákazníkovy vnitřní databáze určenému k zobrazení.

TAI

Backend administrátorské části spočívá pouze v implementaci CRUD operací nad MongoDB databází obsahující data. Tato data popisují způsob zobrazení objektů (souřadnice, velikost, orientace, API klíč, …).

3.3.4. Place Reservations 3

PR je neustálé vyvíjená aplikace sloužící především k rezervaci pracovních a parkovacích míst, disponuje však také mnohými doplňkovými funkcemi, jako například vyhledávání lokací zaměstnanců v budově, správa pracovního rozvrhu zaměstnanců, sledování obsazenosti meetingových místností apod. Základem této aplikace je již zmíněná univerzální mapa integrované TAI služby. PR tedy mimo jiné slouží jako ukázkové využití TAI. Architektura PR je totožná s architekturou TAI.

Pro zobrazení relevantních dat přistupuje PR k mnoha webovým službám. Díky spojení s těmito službami dokáže aplikace v současné verzi na mapě zobrazit:

- obsazenost pracovních a parkovacích míst v libovolném čase
- obsazenost a vybavení meetingových místností
- data z teplotních senzorů rozmístěných po budově
- pozici vybraných zaměstnanců v reálném čase (Jedná se o testovací provoz služby IOT oddělení. Vybraní zaměstnanci nosí bluetooth náramky, které živě přenáší jejich pozici.)
- umístění bezpečnostních prvků (hasicí přístroje, požární hlásiče, únikové východy, …)

PR rovněž dočasně obsahuje administrátorskou část TAI, jejíž vývoj stále probíhá. Po dokončení by měla být přesunuta do samostatné aplikace.

Dashboard

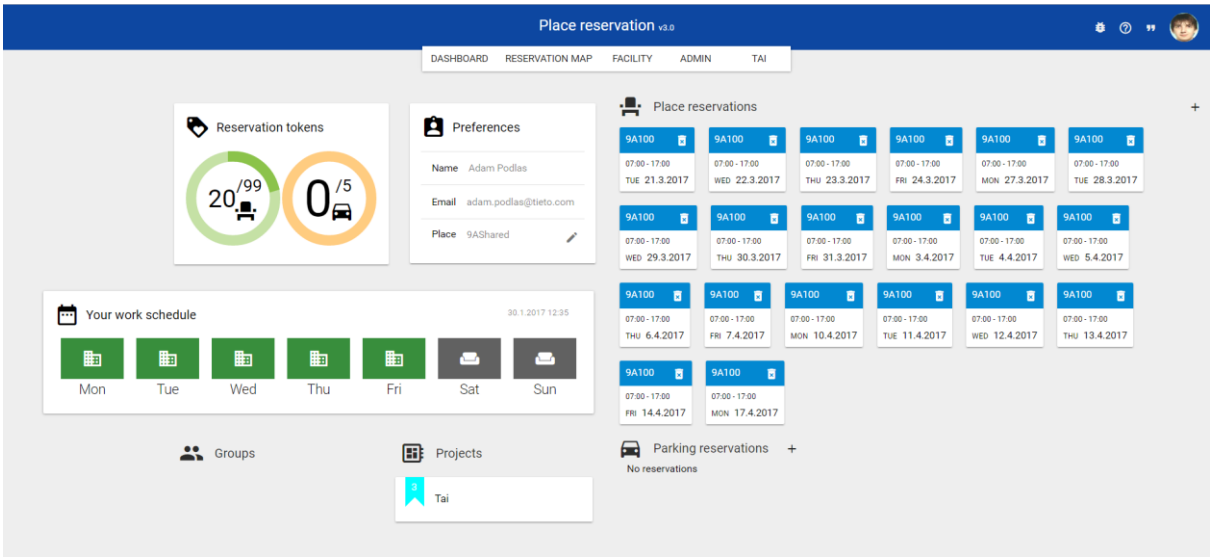
Dashboard je úvodní stránka aplikace PR. Zde má uživatel přehled o všech podstatných datech. Vzhled celé aplikace se řídí pravidly tzv. *Material design*, dashboard se tedy skládá z několika karet obsahujících data seskupená dle kategorie.

Na první kartě nalezneme údaje o aktuálním stavu tokenů uživatele. Existují dva typy tokenů. Jeden pro rezervaci pracovního a druhý parkovacího místa. Jedna rezervace stojí jeden příslušný token. Po skončení rezervace se token vrátí na účet uživatele. Každý uživatel tedy může mít v jednu chvíli maximálně tolik rezervací, kolik má tokenů.

Vedle karty tokenů se nachází karta s údaji o uživateli – jméno, email, pracovní místo. Pole pro pracovní místo je editovatelné, ostatní data jsou pouze pro čtení. Jako pracovní místo si uživatel zadá číslo svého stolu, případně pokud nemá stálé místo, má možnost zadat patro, na kterém obvykle rezervuje.

Další karta slouží pro nastavení pracovního rozvrhu uživatele. Karta obsahuje jedno tří stavové pole pro každý den v týdnu. Uživatel zde zadává, zda bude daný den pracovat ve firmě, z domu, nebo vůbec. Manažeři si poté tyto rozvrhy svých podřízených mohou prohlédnout v manažerské sekci PR použít tyto údaje k plánování.

Dále dashboard obsahuje seznam aktuálních rezervací uživatele a možnost rychlého vytvoření rezervace bez nutnosti hledat místo na mapě. Při vytváření rychlé rezervace se berou v úvahu uživatelovy předchozí rezervace a formulář se podle nich předvyplní.



Obrázek 7 Place Reservations 3: Dashboard

Ostatní stránky

Další stránky aplikace již poskytují specifické funkce. Pro běžné uživatele je přístupná krmě dashboard pouze stránka s mapou popsaná výše. Pro manažery je zpřístupněna stránka se seznamem jeho podřízených. Tento seznam obsahuje u každého zaměstnance jeho aktuální místo a pracovní rozvrh. Následuje facility stránka viditelná pouze pro zaměstnance s příslušným oprávněním, která slouží pro správu míst a skupin. Zde je možné měnit vlastníka místa, případně označit místo jako sdílené a umožnit tak jeho rezervování. Dále se zde nachází vytváření a editace skupin, které slouží k zpřístupnění určitých míst pouze vybraným zaměstnancům.

Joyride

V období mého působení ve firmě jsem měl možnost přijit do styku se spoustou feedbacku uživatelů. Jelikož byla většina negativních ohlasů způsobena neznalostí ovládání, nebo fungování nové aplikace, rozhodl jsem se do nové verze PR implementovat komplexní system průvodce stránkou pro nové uživatele.

Jako základ této funkcionality jsem použil npm modul react-joyride, který poskytuje funkcionalitu průvodce. Joyride funguje velmi jednoduše. Pro definici právě zobrazeného objektu používá *step.* V případě předání pole stepů je zobrazuje postupně, poté co uživatel klikne na další. Aby Joyride věděl, kde má daný step zobrazit, je třeba stepu definovat položku *selector*. Do této položky je třeba zadat DOM selector prvku, u kterého se má step zobrazit.

```
{
  title: 'Welcome to Place Reservation 3.0!',
  text: 'Let\'s take a little tour through the application.',
  selector: `.${JOYRIDE_WELCOME}`,
  position: 'bottom',
  type: 'hover',
  added: '2017-01-19T00:00:00',
},
```

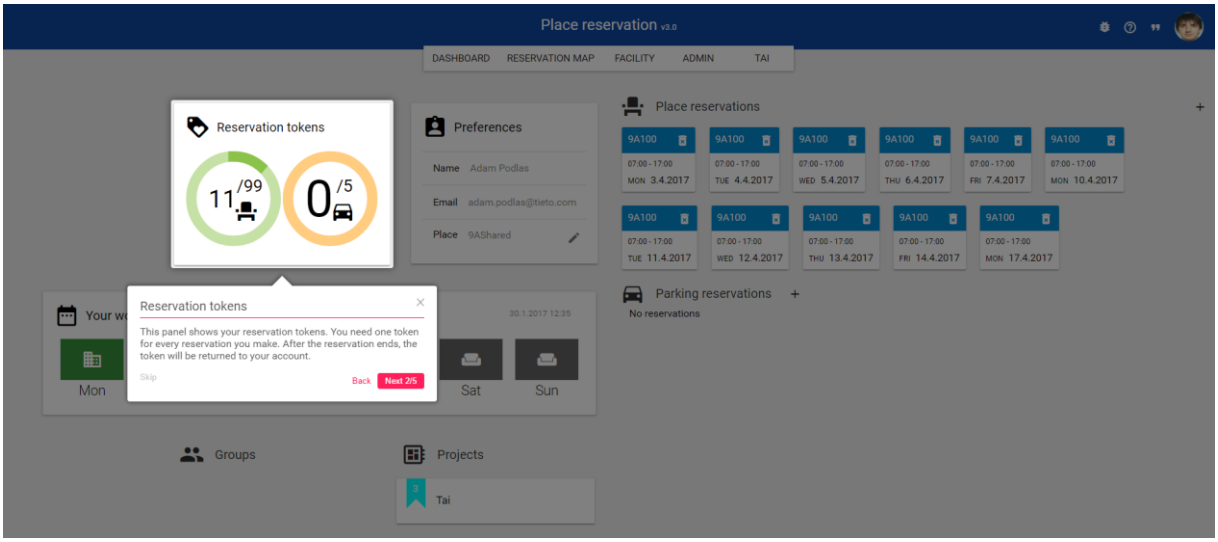
Kód 5 Ukázka syntaxe Joyride stepu

Jelikož jsem chtěl zachovat nezávislost mezi našimi react komponenty, vytvořil jsem pro Joyride samostatný kontejner v Redux store, a jak zapnutí průvodce, tak přepínání stepů jsem řešil pomocí Redux Observable. Stepy jsem definoval předem v samostatném souboru pro konstanty. Pro každou stránku jsem definoval jedno pole stepů. Poté už stačilo jen vytvořit stream z požadovaných akcí (např. přechod na danou stránku, nebo otevření dialogu) a definovat, které pole stepů se má po této akci spustit.

```
export const joyrideDialogTabEpic = (action$) =>
  action$.ofType(DIALOG_CHANGE_TAB)
    .map(action => action.tab ?
        STEPS_CREATE_RESERVATION_ADVANCED
        :
        STEPS_CREATE_RESERVATION
    )
    .flatMap((steps) => [{
      type: JOYRIDE_ACTION_SET_STEPS,
      steps,
    }, {
      type: JOYRIDE_ACTION_SET_READY,
      ready: true,
    }]);
```

Kód 6 Metoda, která po akci na otevření dialogu spustí přísušné pole stepů

Jediná změna, která se tedy musí provést v cizích komponentách, u kterých chci průvodce zobrazit, je přidání css class danému elementu. Komponenta zůstane nezávislá na Joyride kontejneru, protože v jeho nepřítomnosti bude pouze obsahovat css class, která nebude nikde definována, což není na závadu.



Obrázek 8 Joyride v aplikaci Place Reservations 3

Na backendu je tato funkcionalita řešená pouze jedním atributem v tabulce uživatelských profilů. Typ atributu je *Timestamp* – tedy datuma čas, odeslaný z frontendu. Joyride step se zobrazí pouze tehdy, je-li datum zadané v jeho položce *added* pozdější, než datum v profile uživatele. Tím mám zaručeno, že mohu do budoucna přidávat nové stepy například pro nové funkcionality aplikace, a uživatelům se zobrazí k ním definované stepy, bez nutnosti procházet znovu celého průvodce. [8]

Ukázka integrace TAI

Následuje ukázka použití modulu TAIPLATE pro reálné použití. Jako příklad uvádím zobrazení meetingové místnosti na mapě v PR.

Vytvoření vrstvy elementu v TAI editoru

Prvním krokem je vytvořit vrstvu a odpovídající elementy v editoru TAI. Spolu se souřadnicemi, rozměry a dalšími parametry je důležité zadat parametr *element key*. Pomocí tohoto parametru pak na servisní vrstvě TAIPLATE dojde ke správnému přiřazení dat k tomuto elementu. V případě našich meetingových místností použijeme název místnosti, o kterém víme, že je unikátní.

Implementace spojení dat na servisní vrstvě

Po vytvoření vrstvy je třeba implementovat metodu spojující data TAI a PR. Vstupními argumenty této metody jsou: vrstva z TAI modulu, seznam meetingových místností, a časový interval. Tento interval určuje čas, pro který zjišťujeme obsazenost místností z cizí aplikace skrze její REST API.

Metoda nejprve natáhne data ze služby poskytované jiným oddělením firmy. Poté prochází v paralelním streamu jednotlivé elementy mapové vrstvy a každému z nich přiřadí příslušná data. Dále se specifikuje momentální dostupnost místnosti pomocí porovnávání aktuálního času s časy rezervací.

```
private Optional<DataLayerDTO> mergeByElementKeyAndMeetingRoomNumber(
    DataLayerDTO source,
    List<MeetingRoomDTO> data,
    LocalDate from, LocalDate to)
{
   if (source == null)
      return Optional.empty();

   List<ResourceFreeBusyDTO> resource = meetingRoomConnector.fetchRoomStatus(
         from,
         to,
         data.stream()
            .map(MeetingRoomDTO::getOriginId)
            .collect(Collectors.toList())
    );


   //Merge data by the place number and element key
   source.getElements()
         .stream()
         .parallel()
         .forEach(item -> {
            // Find data by element key and set the result to the data
            Optional<MeetingRoomDTO> temp = data
                  .stream()
                  .filter(e ->
                    e.getPlace().getPlaceNumber().equals(item.getElementKey()))
                  .findFirst();
            if (temp.isPresent()) {
               //find reservations from the resource
               resource
                     .stream()
                     .filter(res ->
                       res.getResourceId().equals(temp.get().getOriginId()))
                     .findFirst()
                     .ifPresent(resourceFreeBusyDTO ->
                       temp.get().setReservations(
                           resourceFreeBusyDTO.getReservations()
                                 .stream()
                                 .map(reservationMapper::map)
                                 .collect(Collectors.toList())
                           )
                     );
               //set availability
               boolean available = temp.get().getReservations() != null &&
                                   !(temp.get().getReservations()
                     .stream()
                     .filter(reservation ->
                      reservation.getStart() != null && reservation.getEnd() !=
                      null)
                     .filter(reservation ->
                          reservation.getStart()
                          .isBefore(LocalDateTime.now()) &&
                           reservation.getEnd().isAfter(LocalDateTime.now())
                     ).count() > 0);
               temp.get().setIsAvailable(available);
               //set data
               item.setData(temp.get());
   return Optional.of(source);
}
```

Kód 7 Metoda spojující data PR a TAI

Vytvoření vlastní komponenty pro zobrazení

Abychom na frontendu mohli tato data efektivně zobrazit, je ideální řešení vytvořit si vlastní ReactJS komponentu. V té si můžeme nadefinovat libovolný vzhled a chování. Takovýchto komponent si můžeme vytvořit libovolný počet a při integraci s TAIPLATE npm modulem je pouze předáme pomocí *props*. TAIPLATE modul přijímá vlastní komponenty ve formátu mapy, kde klíč tvoří typ elementu a hodnota je samotná komponenta. Aby došlo ke správnému přiřazení mapového elementu ke komponentě, musí mít tento element v editoru nastaven typ odpovídající klíči v mapě.

3.4. Utilizace vývojového prostředí a práce s Javascriptem

Jednou z největších výhod vývoje v Javascriptu je, že programátorovi umožní v podstatě cokoliv. Tato vlastnost však zároveň představuje i jeho největší nevýhodu. V praxi to znamená, že pokud Javascriptový kód píše člověk bez ohledu na to, jakým způsobem se s ním bude nadále pracovat, nebo prostě jen někdo kdo bere Javascript jen jako „nutné zlo“, je extrémně časově náročné, někdy až téměř nemožné, se v tomto kódu vyznat a používat, či udržovat jej.

Postupem času jsem začal zjišťovat, že spousta zažitých nevýhod vývoje v Javascriptu se dá minimálně částečně odstranit a že se více než vyplatí investovat čas do utilizování procesu vývoje, které může přinést velmi výrazné urychlení jak vývoje samotného, tak údržby a začleňování dalších lidí do procesu. Nejvýznamnější poznatky rozepisuji v následujících podkapitolách. Jako vývojové prostředí používám WebStorm od Jetbrains, pro jiné prostředí se tedy mohou některé věci lišit.

3.4.1. Našeptávač

Ačkoliv jsou Jetbrains prostředí známá velmi propracovanou indexací a s tím spojeným našeptávačem, narazí tato funkcionalita na problém, jakmile začnete používat populární knihovnu Immutable.js. Ta totiž používá místo standartního JS objektu mapu, proto je nutné k hodnotám atributů přistupovat pomocí volání metody *get()* a zasláním klíče jako argumentu. Samozřejmě že IDE neví, jaké klíče daná mapa obsahuje, je tedy nemožné použít našeptávač.

Další problém je specifický pro knihovnu ReactJS. Jak jsem již zmínil, data se mezi React komponentami předávají pomocí *props*. Uvnitř samotných komponent tedy nemáme možnost vidět, jaký tvar mají data, která nám byla předána. Toto lze vyřešit důkladným definováním pomocí tzv. *PropTypes. PropTypes* se definují pomocí JS objektu. K názvu každé položky *props* se definuje, jakého je typu. V případě Immutable.js mapy je možno použít *ImmutablePropTypes.* To nám umožní kontrolovat jaké klíče a hodnoty obsahuje daná mapa.

V praxi se nám tedy pro minimalizaci těchto problémů osvědčilo definování modelu pomocí konstant a *propTypes*. Pro každý model vytvoříme soubor, který exportuje objekt s konstantami, metodu pro mapování a objekt obsahující *propTypes.* Konstanty odpovídají názvům atributů modelů a umožňují funkčnost našeptávače pří použití Immutable.js map a jejich metody get(). Dále konstanty řeší změnu názvu atributu modelu, který je přijímán například přes REST API – stačí změnit hodnotu konstanty. Funkce pro mapování pouze namapuje objekt na Immutable.js mapu a tím zaručí, že má

správný tvar. Objekt obsahující *propTypes* použijeme, v definici *propTypes* komponenty, která bude tento model přijímat.

3.4.2. Zpřehlednění kódu

Jako první nástroj pro udržování přehledné syntaxe jsem používal ESlint. Ten sice poskytoval široké možnosti konfigurace a kvalitní integraci s WebStorm prostředím, ale většinu problémů nedokázal vyřešit automaticky. Ruční řešení problémů samozřejmě stálo nějaký čas a hlavně nervy. Začal jsem tedy hledat více automatizovaný nástroj.

Druhý pokus jsem věnoval nástroji Prettier dostupnému jako npm modul. Ten na rozdíl od ESlintu neoznačuje problémová místa v kódu, ale celý kód přeloží na abstraktní syntaktický strom, a jeho zformátovanou verzí nahradí původní kód. Zmíněný přístup umožňuje komplexnější formátování. Oproti ESlintu například dokáže přepsat syntaxi příliš dlouhého řádku, nebo správně naformátovat úseky kódu obsahující řetězení metod.

Prettier sice nemá přímou integraci s WebStorm prostředím, ale přesto ho lze relativně snadno nakonfigurovat a z prostředí spouštět. Je připraven na spouštění z CLI a veškeré konfigurace lze provést příslušnými parametry. WebStorm poskytuje velmi dobrou podporu pro spouštění scriptů třetích stran, stačí v nastavení přidat nový *External tool*. Při editaci zadáme cestu ke globálně nainstalovanému Prettier npm modulu a zadáme požadované parametry spuštění. Jako jeden z parametrů lze nastavit soubor, nebo adresář pro který se má Prettier spustit. Ten lze zadat jako proměnnou, v mém případě jsem kvůli rychlosti spuštění nastavil cestu k aktuálnímu souboru. Dále je třeba nastavit cestu k projektu. Díky použití proměnných, je nyní Prettier nastaven globálně a funguje pro jakýkoliv aktuálně otevřený projekt. Dále je možné si přiřadit spuštění external toolu na libovolnou klávesovou zkratku a tím proces ještě zrychlit. Po správném nastavení je zpřehlednění kódu jednoho souboru otázkou asi dvou sekund.

3.4.3. Ladění

Při přechodu z jazyků jako Java, nebo C#, je významným problémem ladění. Například Google Chrome sice poskytuje kvalitní možnost ladění Javascriptu, ale problém nastává, když se už do prohlížeče dostane jiný kód, než vývojář píše, což je v dnešní době běžné. Ať už v důsledku přeložení do starší verze EcmaScript, použití bundlovacího nástroje jako webpack, nebo třeba obfuskace.

Pro tento problém jsem našel řešení, umožňující debug aplikací přímo ve WebStormu, téměř k nerozeznání od ladění Java aplikací v IntelliJ IDEA. Jako první jsem musel přidat do konfiguračního souboru webpacku položku devtool s hodnotou "source-map". Ta způsobí, že webpack k výslednému balíčku poskytne soubor obsahující mapování k původním souborům. Dále jsem ve webstormu přidal novou spouštěcí konfiguraci „JavaScript Debug“ a zadal k ní adresu na které běží vyvíjená aplikace. Po spuštění této konfigurace se zobrazí výstupní soubory webpacku. Mezi nimi jsem našel složku odpovídající „app/src“ (rodičovská složka obsahující veškeré zdrojové kódy) a přiřadil k ní příslušnou složku z projektu. Po restartu byl WebStorm schopen spustit a řídit ladění aplikace přímo z prostředí.

4. Uplatněné a chybějící znalosti

Z ve škole nabytých znalostí jsem uplatnil například základy jazyka Java (i když na významné novinky verze 1.8 bohužel nepřišla ve škole řeč). Dále mi byly velmi užitečné znalosti z oblasti relačních databází, a to jak z hlediska návrhu, tak dotazovacího jazyka SQL. V neposlední řadě mi přišly vhod alespoň elementární vědomosti o metodice scrum a agilním vývoji obecně.

Ohledně chybějících znalostí bych ze strany školy ocenil především větší přehled o prakticky používaných nástrojích a technikách. Osobně si myslím, že důležitější, než hluboká znalost jednoho jazyka/technologie, je co nejširší záběr. Před nástupem na praxi jsem neměl tušení o existenci NoSQL, Git, Maven, Spring, Javascript, Node.js, nebo třeba serverless architektury. Nečekám, že si ze školy odnesu detailní znalosti v těchto oblastech, ale vzhledem k tomu, že se jedná o aktuální a momentálně velmi rozšířené technologie, bylo by myslím vhodné, aby o těchto technologiích měl absolvent povědomí a věděl, k čemu se používají. Dále bych uvítal rozšíření vyučované oblasti programovacích paradigmat například o *event driven* a *reactive,* případně dalších.

5. Závěr

Celkově považuji volbu absolvování odborné praxe jako velmi přínosnou. Z hlediska získaných vědomostí a zkušeností mi tato relativně krátká doba dala více než zbylá doba studia. Přiblížila se mi představa o řešení reálných problémů. V průběhu práce na reálných projektech jsem se setkal s problémy jen těžko simulovatelnými ve školním prostředí (např. nejasná zadání od zákazníků, kteří sami nevěděli, co chtějí, nebo třeba hledání kompromisů mezi našim a cizím řešením při integraci projektů různých vývojových týmů).

Práce v prostředí firmy mi také pomohla uvědomit si význam technik, které nemají vliv na samotné řešení, ale slouží pouze jako podpora vývojářům a vůbec lidem pohybujícím se kolem projektu. Ať už jde o pluginy do IDE pro zpřehlednění kódu, nástroje pro automatizaci deploy procesu, nebo jen domluvené zásady psaní kódu mezi členy týmu. Všechny tyto nástroje se mi dříve jevily jako nevýznamné detaily. Po vyzkoušení práce ve firmě jsem si však uvědomil, jak zásadně mohou ovlivnit dobu práce na projektu a s tím samozřejmě i jeho cenu.

6. Literatura

[1] Informace o Tieto [online]. Tieto s.r.o. [cit. 2017-03-31]. Dostupné z: https://www.tieto.cz/tieto-o-nas/

[2] What is MongoDB? [online]. MongoDB, Inc. [cit. 2017-03-31]. Dostupné z: https://www.mongodb.com/what-is-mongodb/

[3] About Node.js [online]. Node.js Foundation. [cit. 2017-03-31]. Dostupné z: https://nodejs.org/en/about/

[4] React. [online]. Facebook Inc. [cit. 2017-03-31]. Dostupné z: https://facebook.github.io/react/

[5] Redux - Introduction [online]. [cit. 2017-03-31]. Dostupné z: http://redux.js.org/docs/introduction/

[6] Redux Observable - Introduction [online]. [cit. 2017-03-31]. Dostupné z: https://redux-observable.js.org/

[7] SVG [online]. Mozilla Developer Network. [cit. 2017-03-31]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/SVG/

[8] React-joyride [online]. [cit. 2017-03-31]. Dostupné z: https://github.com/gilbarbara/react-joyride/